C++ Basics

Dr. Xiaolan Zhang
Fordham University

```
/* This is a program that simply prints out Hello world
   to the terminal, and move cursor to next line
   By X. Zhang,
*/

#include <iostream>          //This is preprocessor directive

int main()
{
   std::cout <<"Hello world\n";  //display to terminal window
                           // this is a blank line, added here for readability
   return 0;                  //program exits, return 0 to indicate success
}
```

Curly braces enclose the body of function

```
/* This is a program that simply prints out Hello world
   to the terminal, and move cursor to next line
   By X. Zhang,
*/

#include <iostream>          //This is preprocessor directive
using namespace std;  // In order to use cin, cout etc declared in std

int main()
{
    cout <<"Hello world\n";  //display to terminal window
                           // this is a blank line, added here for readability
    return 0;                //program exits, return 0 to indicate success
}
```

Curly braces enclose the body of function

Keyword

- In previous example, the following words are keywords of C++
  - int, main, return, using …

- Keywords (also called reserved words)
  - Are used by the C++ language
  - Must be used as they are defined in the programming language
  - Cannot be used for other purposes

Special characters in C++

- # is used to start a directive
- ; is used to end an statement
- // is used to start a single line of comments
- /* and */ are used to enclose multi-line comments
- Double quotation marks " used to enclose a string constant
  - Note that this is not the same as “”

- Single quotation marks used to enclose a character constant

- *Important to use plain text editor to edit your program: not word, WordEdit, as they introduce other characters, and convert quotations marks…*

cout statement

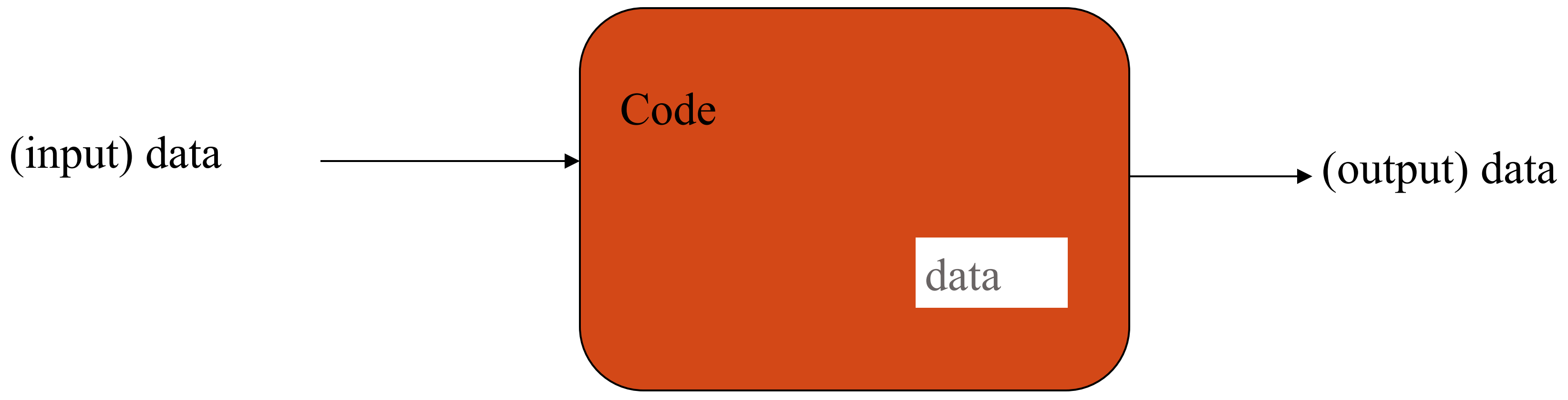
cout <<"Hello world\n";

- cout: pronounced as see-out, stands for character output
  - Defined in iostream, a header file (a source code) that is part of C++ standard library
  - Represents the terminal window that the program is running from
- << insertion operator: to insert (display) something in terminal window
  - Can display multiple values in single statement, e.g.,
  - cout <<"Hello world, " << "this is my first program!\n";
  - cout <<"Hello world, "

    << “this is my first program!\n";

One statement can be split into multiple lines.

Computation



- Input: from keyboard, files, other input devices, other programs, other parts of a program
- Computation – what our program will do with the input to produce the output.
- Output: to screen, files, other output devices, other programs, other parts of a program

Program structure

Single batch of input

1 Read inputs

2 Computation

3 Write output

Multiple batch of input

1 Read inputs

2 Computation

3 Write output

4 Go back to 1

Input and Output

```
// read first name:
#include <iostream>
#include <string>
using namespace std;

int main()
{
    cout << "Please enter your first name (followed " << "by 'enter'):\n";
    string first_name;
    cin >> first_name;
    cout << "Hello, " << first_name << '\n';
}

// note how several values can be output by a single statement
// a statement that introduces a variable is called a declaration
// a variable holds a value of a specified type
// the final return 0; is optional in main()
// but you may need to include it to pacify your compiler
```

Input and Type

- We read data/or input into a variable
  - Here, **first_name**
- A variable has a type
  - Here, **string**
  - The type of a variable determines what operations we can do on it
- Here, **cin>>first_name;** reads characters until a whitespace character is seen
  - White space: space, tab, newline, …

Overview

- Variables and Assignments
- Data Types and Expressions
- Input and Output
- Program Style

C++ Variables

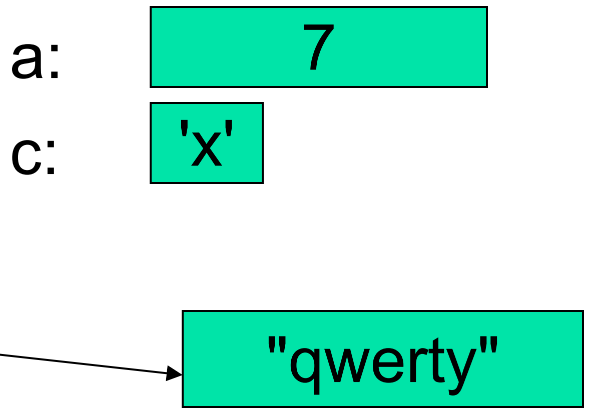
- Variables are like small blackboards
  - We can write a number on them
  - We can change the number
  - We can erase the number
- C++ variables are memory locations
  - We can write a value in them
  - We can change the value stored there
  - We cannot erase the memory location
    - Some value is always there

C++ Variables

- A variable is some memory that can hold a value of a given type
- A **variable** has a name
  - Each variable has name, type, value

A **declaration** names a variable

```
int a = 7;
char c = 'x';
string s = "qwerty";
```



Size of memory varies for objects of different types !

A C++ Program (*part 1 of 2*)

```
#include <iostream>
using namespace std;
int main( )
{
    int number_of_bars;
    double one_weight, total_weight;

    cout << "Enter the number of candy bars in a package\n";
    cout << "and the weight in ounces of one candy bar.\n";
    cout << "Then press return.\n";
    cin >> number_of_bars;
    cin >> one_weight;

    total_weight = one_weight * number_of_bars;

    cout << number_of_bars << " candy bars\n";
    cout << one_weight << " ounces each\n";
    cout << "Total weight is " << total_weight << " ounces.\n";

    cout << "Try another brand.\n";
    cout << "Enter the number of candy bars in a package\n";
    cout << "and the weight in ounces of one candy bar.\n";
    cout << "Then press return.\n";
    cin >> number_of_bars;
    cin >> one_weight;

    total_weight = one_weight * number_of_bars;

    cout << number_of_bars << " candy bars\n";
    cout << one_weight << " ounces each\n";
    cout << "Total weight is " << total_weight << " ounces.\n";

    cout << "Perhaps an apple would be healthier.\n";

    return 0;
}
```

Display 2.1 (1/2)

Display 2.1 (2 /2)

A C++ Program (*part 2 of 2*)

Sample Dialogue

```
Enter the number of candy bars in a package
and the weight in ounces of one candy bar.
Then press return.
11 2.1
11 candy bars
2.1 ounces each
Total weight is 23.1 ounces.
Try another brand.
Enter the number of candy bars in a package
and the weight in ounces of one candy bar.
Then press return.
12 1.8
12 candy bars
1.8 ounces each
Total weight is 21.6 ounces.
Perhaps an apple would be healthier.
```

Identifiers

- Variables names are called **identifiers**
- Choosing variable names
  - Use meaningful names that represent data to be stored
  - First character must be
    - a letter or the underscore character, _
  - Remaining characters must be
    - Letters, numbers, underscore character
- Keywords, such as return, main, if, … cannot be used as identifiers

Declaring Variables

- Before use, variables must be declared
- Declaration syntax:
  type_name variable_1 , variable_2, . . . ;

  - Tells the compiler: I need variable(s) named … to store .. type of data

```
int        number_of_bars;
double one_weight,  total_weight;
```

Try this: what does the above three declarations say?

How about this? char   response, option;

Two locations for variable declarations

Immediately prior to use

```
int main()
{
   …
   int sum;
   sum = score1 + score 2;
   …
   return 0;
}
```

- **At the beginning**

```
int main()
{
     int sum;
    …
    sum = score1 +
score2;
     …
 return 0;
}
```

Assignment Statements

- **An assignment statement** changes the value of a variable
  - total_weight = one_weight + number_of_bars;
    - total_weight is set to the sum one_weight + number_of_bars
  - Assignment statements end with a semi-colon
  - **Left hand side (LHS):** variable whose value is to be changed
  - **Right hand side (RHS):** new value for the LHS variable:
    - Constants --   age = 21;
    - Variables --   my_cost = your_cost;
    - Expressions -- circumference = diameter * 3.14159;

Assignment Statements

- The '=' operator in C++ is not an equal sign
  - The following statement cannot be true in algebra
    - number_of_bars = number_of_bars + 3;
  - In C++ it means the new value of number_of_bars is the previous value of number_of_bars plus 3

Initializing Variables

- Declaring a variable does not give it a value
  - Giving a variable its first value is initializing the variable
- Variables are initialized in assignment statements

```
double mpg;        // declare the variable
mpg = 26.3;         // initialize the variable
```

- Declaration and initialization can be combined
  - Method 1

```
double mpg = 26.3, area = 0.0 , volume;
```

  - Method 2

```
double mpg(26.3),  area(0.0), volume;
```

Exercises

- Can you
  - Declare and initialize two integers variables to zero? The variables are named feet and inches.

  - Declare and initialize two variables, one int and one double? Both should be initialized to the appropriate form of 5.

  - Give good variable names for identifiers to store
    - the speed of an automobile?
    - an hourly pay rate?
    - the highest score on an exam?

Overview

- Variables and Assignments
- Data Types and Expressions
- Input and Output
- Program Style

Types

- C++ provides a set of types
  - E.g. **bool**, **char, int, double**
  - **Called "built-in types"**
- C++ programmers can define new types
  - Called "user-defined types"
  - We'll get to that eventually, mostly in CS2
- C++ standard library provides a set of types
  - E.g. **string**, **vector**, **complex**
  - Technically, these are user-defined types
    - they are built using only facilities available to every user

Builtin Types (1)

- Boolean type represents value of true or false
  - **bool**
  - ex: bool invalidInput; // used to mark invalid input

- Character type represents a single character, such as q, a, B, \n, (, …
  - **char**
  - **Ex: char choice = ‘q’;**

Builtin Types (2):Integer Number types

- Integer numbers (int, short,long) are whole numbers without a fractional part
  - Includes zero and negative numbers
  - Used for storing values that are conceptually whole numbers (e.g. pennies)
  - Process faster and require less storage space (compared to floating-point numbers)

Types and literals

int pennies = 8;

cout << " Hello world\n ";

- Literal constants: values that occurs in the program
  - Literal, as we can only speak of it in terms of its value
  - Constant: its value cannot be changed
- How to write literals?
  - Depending on the type of the literal
  - 8 is of type int, “Hello world\n” is of type string
- More examples:
  - bool validInput = true;
  - bool continue = false; // reserved words
  - Character literals: **'a', 'x', '4', '\n', '$'.**
  - Integer literals: **0, 1, 123, -6, 0x34, 0xa3, 024**
  - Floating point literals: **1.2, 13.345, .3, -0.54, 1.2e3, . 3F, .3F**
  - String literals: **"asdf"**, **"Howdy, all y'all!“**

Builtin Types(3): Floating point types

- Floating-point types represents number with decimal points, such as 3.14
  - **double,** and **float**
  - **Process slower and require more storage space**

Type char

- **char:** can be any single character from the keyboard
- To declare a variable of type char:

  char letter;
- **Character constants** are enclosed in single quotes

  char letter = 'a';
- **Strings constant**, even if contain only one character, is enclosed in double quotes
  - cout << "Hello world ";
  - "a" is a string of characters containing one character
  - 'a' is a value of type character

The type *char*

```
#include <iostream>
using namespace std;
int main( )
{
    char symbol1, symbol2, symbol3;

    cout << "Enter two initials, without any periods:\n";
    cin >> symbol1 >> symbol2;

    cout << "The two initials are:\n";
    cout << symbol1 << symbol2 << endl;

    cout << "Once more with a space:\n";
    symbol3 = ' ';
    cout << symbol1 << symbol3 << symbol2 << endl;

    cout << "That's all.";

    return 0;
}
```

Sample Dialogue

```
Enter two initials, without any periods:
J B
The two initials are:
JB
Once more with a space:
J B
That's all.
```

More on floating point constants

- Simple form must include a decimal point
  - e.g., 34.1   23.0034   1.0   89.9
- Scientific Notation form
  - e.g. 3.41e1 means 34.1

    3.67e17 means 367000000000000000.0

    5.89e-6 means 0.00000589
  - Number left of e does not require a decimal point
  - Exponent cannot contain a decimal point

C++ Standard Library Type: string

- **string** is a class, different from primitive data types discussed so far
  - Requires the following be added to the top of your program:
    ```
    #include <string>
    ```
  - Use double quotes around the text to store into the string variable
- To declare a variable of type string:

```
string name = "Apu Nahasapeemapetilon";
```

DISPLAY 2.4 The string class

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
    string middle_name, pet_name;
    string alter_ego_name;

    cout << "Enter your middle name and the name of your pet.\n";
    cin >> middle_name;
    cin >> pet_name;

    alter_ego_name = pet_name + " " + middle_name;

    cout << "The name of your alter ego is ";
    cout << alter_ego_name << "." << endl;

    return 0;

{
```

Sample Dialogue 1

```
Enter your middle name and the name of your pet.
Parker Pippen
The name of your alter ego is Pippen Parker.
```

Sample Dialogue 2

```
Enter your middle name and the name of your pet.
Parker
Mr. Bojangles
The name of your alter ego is Mr. Parker.
```

Overview

- Variables and Assignments
- Data Types
- Expressions
- Input and Output
- Program Style

Operation on data

- Once we have variables and constants, we can begin to operate with them.
- C++ defines operators.
- Operators in C++ are mostly made of signs that are not part of alphabet but are available in all keyboards.
  - Shorter C++ code and more international

Operators

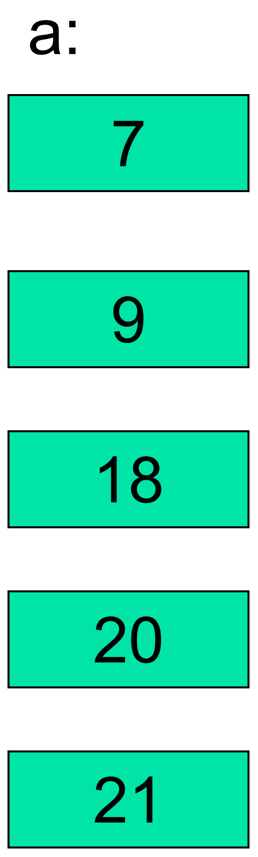
- **Assignment (=)**
  - The assignment operator assigns a value to a variable.
- **Arithmetic operators (+, -, *, /, %)**
  - five arithmetical operations supported by the C++ language are
    - Addition: +
    - subtraction: -
    - Multiplication: *
    - Division: /
    - Modulo: %, gives remainder of a division of two values.
      a = 11 % 3; // a will contain the value of 2

Assignment and increment

```
// changing the value of a variable
int a = 7;         // a variable of type int called a
                   // initialized to the integer value 7
a = 9;          // assignment: now change a's value to 9

a = a+a;          // assignment: now double a's value

a += 2;           // increment a's value by 2
             // a shorthand notation for a = a+2;
++a;              // increment a's value (by 1)
             //shorthand notation for a = a+1;
```



Simple arithmetic

```
// do a bit of very simple arithmetic:
#include <math.h>

int main()
{
    cout << "please enter a floating-point number: "; // prompt for a number
    double n;                                              // floating-point variable
    cin >> n;
    cout << "n == " << n
            << "\nn+1 == " << n+1                        // '\n' means "a newline"
            << "\nthree times n == " << 3*n
            << "\ntwice n == " << n+n
            << "\nn squared == " << n*n
            << "\nhalf of n == " << n/2
            << "\nsquare root of n == " << sqrt(n)    // library function
            << endl;                                     // another name for newline
}
```

If the user enters 25 upon the prompt, what's the output?

Arithmetic Expressions

- Arithmetic operators can be used with any numeric type, i.e., operand can be any numeric type
- Result of an operator depends on types of operands
    - If both operands are int, result is int
    - If one or both operands are double, result is double

Division of Doubles

- Division with at least one double operand produces expected results

```
double divisor, dividend, quotient;
divisor = 3.0;
dividend = 5.0;
quotient = dividend / divisor;
 result: quotient = 1.6666…
```

  - Result is same if either dividend or divisor is of type int

Division of Integers

- int / int produces an integer result

```
int dividend, divisor, quotient;
dividend = 5;
divisor = 3;
quotient = dividend / divisor;
```

- The value of quotient is 1, not 1.666…
- Integer division does not round result, fractional part is discarded!

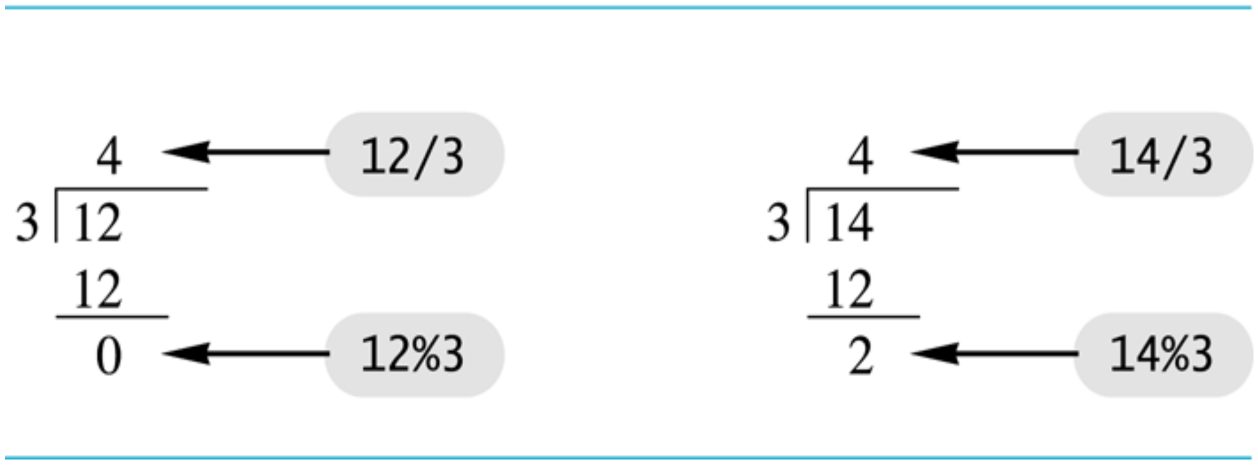
Integer Remainders

- % operator gives remainder from integer division

```
 int dividend, divisor, remainder;
dividend = 5;
divisor = 3;
  remainder = dividend % divisor;
```

The value of remainder is 2

Integer Division



Discussion

- "Giving changes" for Cashier program
  - Instruct the cashier to give changes, e.g., a change of $12.34 should be given in
    - One 10 dollar bill
    - two 1 dollar bills
    - One quarter
    - One nickel
    - Four pennies

Arithmetic Expressions

- Use spacing to make expressions readable
  - Which is easier to read?

    x+y*z   or   x + y * z

- Precedence rules for operators are the same as used in your algebra classes
- Use parentheses to alter the order of operations

  x + y * z   (y is multiplied by z first)

  (x + y) * z   (x and y are added first)

Arithmetic Expressions

Mathematical Formula	C++ Expression
$b^2 - 4ac$	`b*b - 4*a*c`
$x(y + z)$	`x*(y + z)`
$\frac{1}{x^2 + x + 3}$	`1/(x*x + x + 3)`
$\frac{a + b}{c - d}$	`(a + b)/(c - d)`

Operator Shorthand

- Operator shorthand: can be used when applying an arithmetic operation on a variable and saving result back to the varilable,
  - +=

    e.g., count = count + 2; becomes count += 2;
  - *=

    e.g., bonus = bonus * 2; becomes bonus *= 2;
  - /=

    e.g., time = time/rush_factor; becomes time /= rush_factor;
  - %=

    e.g., remainder = remainder % (cnt1+ cnt2); becomes
    remainder %= (cnt1 + cnt2);

Increment/Decrement

- Unary operators require only one operand
  - + in front of a number such as +5
  - - in front of a number such as -5
- **++ increment operator**
  - Adds 1 to value of a variable

```
                     x ++;
is equivalent to     x = x + 1;
```

- **-- decrement operator**
  - Subtracts 1 from value of a variable

```
                     x --;
is equivalent to     x = x – 1;
```

Overview

- ❑ Variables and Assignments
- ❑ Data Types
- ❑ Expressions
- ❑ Input and Output
- ❑ Program Style

Input and Output

- A data stream is a sequence of data: in the form of characters or numbers
- An input stream is data for the program to use, originating from keyboard, a file …
- An output stream is the program's output, destining to monitor, or a file , ..
- Include directives: add library files to our programs
  - To make definitions of the cin and cout available :

    #include <iostream>
- **Using directives:** include a collection of defined names
  - To make names cin and cout available to our program:

    using namespace std;

Output using cout

- **cout** is an output stream for program to send data to monitor
  - insertion operator "<<" inserts data into cout
- cout << number_of_bars << " candy bars\n";
  - sends two items to monitor: value of number_of_bars, and quoted string constant
    - No space added between items, therefore space before the 'c' in candy,
    - A blank space can also be inserted with

      cout << name << " " <<age <<endl ;
  - A new insertion operator is used for each item of output
  - same as

```
                 cout << number_of_bars ;
cout << " candy bars\n";
```

- cout an expression directly

```
cout << "Total cost is $" << (price + tax);
```

Formatting Output

- **Escape sequences:** tell the compiler to treat characters in a special way, allow one to specify special characters
- '\' is escape character
- To create a newline in output use \n, or endl;

```
cout << "Hello\n";
cout << "Hello"<<endl;
```

- Other escape sequences:

```
\t      --  a tab
\\      --  a backslash character
\"      --  a quote character
```

- When printing receipt, use \t to line up different columns
  - Other ways possible …

Formatting Real Numbers

- Real numbers (type double) produce a variety of outputs

```
double price = 78.5;
cout << "The price is $" << price << endl;
```

  - output could be any of these:

```
The price is $78.5
The price is $78.500000
The price is $7.850000e01
```

  - unlikely to get:

```
The price is $78.50
```

Showing Decimal Places

- cout includes tools to specify the output of type double
- To specify fixed point notation
  - setf(ios::fixed)
- To specify that decimal point will always be shown
  - setf(ios::showpoint)
- To specify that two decimal places will always be shown
  - precision(2)
- e.g.:

```
cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(2);
cout    << "The price is $"
        << price << endl;
```

Input Using cin

- cin is an input stream bringing data from the keyboard
- extraction operator (>>) removes data to be used
- e.g.,

```
cout << "Enter the number of bars in a package\n";
cout << " and the weight in ounces of one bar.\n";
cin >> number_of_bars;  // program will wait for input
cin >> one_weight;
```

- code **prompts** user to enter data then reads two data items from cin
  - first value read is stored in number_of_bars
  - second value read is stored in one_weight
  - Data is separated by spaces when entered

Reading Data From cin

- Multiple data items are separated by spaces (space, tab, newline)
    - cin skips blanks and line breaks looking for data
- Data is not read until enter key is pressed
    - Allows user to make corrections

```
cin >> v1 >> v2 >> v3;
```

- Requires three space separated values
- User might type

```
        34  45  12   <enter key>
```

- Or

```
    34  <enter key>
    45 <enter key>
    12   <enter key>
```

Reading Character Data

- following reads two characters but skips any space that might be between

```
char symbol1, symbol2;
cin >> symbol1 >> symbol2;
```

- User normally separate data items by spaces

```
J   D
```

- Results are same if data is not separated by spaces

```
JD
```

Designing Input and Output

- Prompt the user for input that is desired
  - cout statements provide instructions

```
cout << "Enter your age: ";
cin >> age;
```

    - Notice the absence of a new line before using cin
- Echo the input by displaying what was read
  - Gives the user a chance to verify data

```
cout << age << " was entered." << endl;
```

Exercise

- Can you
  - write an input statement to place a value in the variable the_number?
  - Write the output statement to prompt for the value to store in the_number?
  - Write an output statement that produces a newline?
  - Format output of rational numbers to show 4 decimal places?

Overview

- ❑ Variables and Assignments
- ❑ Data Types
- ❑ Expressions
- ❑ Input and Output
- ❑ Constants

Program Style - Constants

- **Literal constants**: have no mnemonic value, i.e., no name
  - total_price = large*14.82+small*12.34;
  - cout << "Hello world";
  - Cons: difficult to find and change when needed, harder to understand
- **Named constants**: give a name to a constant
  - Allow us to change all occurrences simply by changing value of constant
  - e.g.:

    const int WINDOW_COUNT = 10;

    declares a constant named WINDOW_COUNT
  - **const** is keyword to declare a constant
  - Its value cannot be changed by the program (unlike a variable)
  - Common practice: name constants with all capitals

Summary

- New concepts:
  - Variable, identifier, data type
  - Expressions
  - Statements: declaration statements, assignment statements
  - Constant: literal constant and named constant
- We learnt how to
  - Declare a variable
  - Assign value to a variable
  - Input/output: how to read a value from keyboard, how to write to terminal (or, monitor)
  - Write arithmetic expressions